Network Working Group
RFC #671
NIC #31439
December 6,1974

Richard Schantz
BBN-TENEX

RECEIVED
JAN 02 1975

A Note on Reconnection Protocol

INTRODUCTION

This note documents the experience we have had in implementing a modified, experimental version of the Telnet reconnection protocol option within the context of the Resource Sharing Executive (RSEXEC). The reconnection protocol specifies a procedure for transforming a configuration from one in which the initiating process has connections to two correspondent processes, to one in which there is a direct connection between the correspondents. When the procedure is successfully completed, the initiating process is no longer in the communication path between the correspondents.

Resource sharing computer networks and distributed computing will increasingly give rise to specialization by task among the computer installations. In such an environment, a "job" is the dynamically varying interconnection of a subset of these specialized modules. Connections are the "glue" in "bonding" the job together. Reconnection provides for a dynamically changing "bonding" structure. (For a more complete discussion of the utility of reconnection, see RFC 426).

This document deals with reconnection in terms of its current ARPANET definition as a Telnet protocol option. The first section defines a modified reconnection protocol. The second section discusses general network implementation details, while the final section describes aspects of the TENEX/RSEXEC implementation.

Familiarity with the new ARPANET Telnet protocol (RFC 495) is assumed.

I. PROTOCOL for RECONNECTING TELNET COMMUNICATION PATHS

A process initiates the reconnection of two of its Telnet connections by sending (or requesting its "system" to send) the <IAC><DO><RECONNECT> Telnet command sequence over each of the two send connections. The process initiating the reconnection is attempting to cause the direct connection of the objects of the two Telnet connections. In this manner, the initiating process can remove itself from the communication path between Telnet objects.

The initiating process awaits positive responses to both reconnection requests before proceeding further with the reconnection. A reconnection request may be accepted by replying with the Telnet sequence <IAC><WILL><RECONNECT>. It may be rejected by sending the Telnet sequence <IAC><WONT><RECONNECT>. Rejection of both requests means normal communication may resume at once. Rejection of one request (but not the other) requires that the process agreeing to the reconnection be notified by sending it the Telnet sequence <IAC><DONT><RECONNECT> in response to its acceptance reply.

After receiving positive responses to both requests, the initiating agent next selects the object of one of the Telnet connections for a passive role in the subsequent connection attempt. The other is designated as the active participant. The passive participant is to listen on a set of sockets, and the active participant is to send Request for Connections (RFCs) for those sockets. By designating roles, we are trying to reduce the probability of synchronization problems.

The initiating process next enters into subnegotiation with the process designated as being passive. This subnegotiation involves sending the Telnet sequence <IAC> <SB> <RECONNECT> <PASSIVE> <NEWHOST> <NEWSOCKET1> <NEWSOCKET2> <NEWSOCKET3> <NEWSOCKET4> <IAC> <SE>. The <PASSIVE> parameter indicates that the recipient is to listen for RFCs from the socket pair denoted by <NEWHOST> <NEWSOCKET1-4>. The "NEWHOST" is one 8 bit byte designating the address of the host on which the active process (i.e. the one to reconnect to) resides. NEWSOCKET1-4 are four 8 bit bytes indicating the 32 bit send socket number of the Telnet pair from the active process. The <IAC><SE> fields terminate the subnegotiation parameters. The initiating agent awaits a response from the passive process before proceeding. The legal responses are:

1) Telnet sequence <IAC><WONT><RECONNECT>
Meaning: The passive process has decided not to complete the reconnection, after having initially indicated a willingness. This may be due to unexpected parameters during the subnegotiation (e.g. it refuses to connect to NEWHOST), or perhaps some error condition at the passive host.

2) Telnet sequence <IAC><SB><RECONNECT><IAC><SE>
Meaning: Positive acknowledgement of the subnegotiation sequence. The passive process has accepted the reconnection parameters and will proceed with reconnection.

If the reply was <WONT><RECONNECT>, the initiator is obliged to send the Telnet sequence <IAC><DONT><RECONNECT> to the active participant, to cancel the outstanding reconnection request. A confirming <IAC><WONT><RECONNECT> should follow.

The <IAC><SB><RECONNECT><IAC><SE> reply means that the passive participant has begun its connection shutdown, and will listen on the appropriate sockets. The initiator may now close its connections to the passive participant and supply the parameters to the active participant.

This can be done with the assurance that it (the initiator) has done all it can to ensure that the passive process listens before the active process sends its RFCs. Failure to coordinate these actions may result in the failure of the reconnection, if, for example, the passive host does not queue unmatched RFCs. Persistance on the part of the active participant should be an integral part of the protocol, due to uncertainties of synchronization.

The parameter list sent to the active participant is the Telnet sequence <IAC> <SB> <RECONNECT> <ACTIVE> <NEWHOST> <NEWSOCKET1> <NEWSOCKET2> <NEWSOCKET3> <NEWSOCKET4> <IAC> <SE>. The <ACTIVE> parameter indicates to the recipient that it is to send RFCs to the socket pair denoted by <NEWHOST> <NEWSOCKET1-4>. The initiator again waits for a reply. The legal replies are:

1) Telnet sequence <IAC><WONT><RECONNECT>

Meaning: Process will not complete the reconnection (e.g. it couldn´t parse the parameter string).

Possible action of initiator: Attempt to re-establish communication with the passive participant by sending RFCs for the sockets on which the passive participant is listening. This will succeed if the listener is willing to accept connections from either the host/socket specified by the reconnect parameters or the host/socket of the former connection. If it is successful in reestablishing the connection, the initiator could send the Telnet sequence <IAC><DONT><RECONNECT> to confirm that reconnection has been aborted. <SB><RECONNECT><IAC>

unlikely

2) Telnet sequence <IAC><SE>

Meaning: Positive confirmation of the reconnection subnegotiation. The active participant indicates with this reply that it will close the connections to the initiator and send the necessary RFCs to connect to the passive participant. The initiator may close the connections to the active participant, thereby removing itself from the communication path between the objects of the reconnection.

DEFAULT CONDITIONS and RACES

The default for this option is as for most other Telnet options: DONT and WONT. An initiator uses the <DONT><RECONNECT> Telnet sequence to return to the default state, while a participant uses <WONT><RECONNECT> to maintain or return to the default state. The reconnection state is only a transient one. When accepted by all parties, the reconnection state lasts only until the reconnection is completed. Upon completion, and without further interaction among the parties, the state of the new connection is the default state, with the negotiated reconnection forgotten.

Since reconnection is an option concerning the entire Telnet connection, the asynchronous nature of the option processing mechanism exemplified by many other Telnet options (e.g. echo), is not applicable. That is, a race condition occurs when two <IAC><DO><RECONNECT> requests cross each other in the network. A

solution to this problem was presented in RFC 426; the following is a modified version of that protocol extension. The modification is concerned mainly with preserving the right of a process to deny a reconnection attempt by another process, while having its own reconnection request pending.

The race condition is detected when a process receives a <DO><RECONNECT> while awaiting a reply to a <DO><RECONNECT> it has previously issued on the same Telnet connection. (This condition is detected at both ends of the connection). The strategy to resolve the race utilizes a function, evaluated at both ends of the connection, to determine which reconnection request shall take precedence. The evaluation involves comparing the numbers obtained by concatenating the host address (which becomes the high order 8 bits) and the receive socket number (becomes the low order 32 bits) for the two ends of the Telnet connection. The process owning the receive socket with the larger of the two concatenated numbers will have its reconnection attempt preceed that of the other process. Thus, if there is a Telnet connection between host A local sockets X,X+1 and host B local socal sockets Y,Y+1, and if <A><X> is greater than <B><Y>, then the reconnect request from <A><X> must be completed (or aborted) before the reconnection request from <B><Y> can be considered. This is achieved by requiring that the process with the higher <host><socket> number reply to the reconnect request of the other process with an <IAC><WONT><RECONNECT>, thereby cancelling (temporarily) the reconnection attempted from the lower numbered <host><socket>. Since the request emanating from the higher <host><socket> process is given precedence, the process with the lower <host><socket> can reply to the reconnection request as if it had not issued a reconnection request of its own. That is, it may reply <IAC><WILL><RECONNECT> to accept the reconnection attempt, or <IAC><WONT><RECONNECT> to refuse the attempt. This process should note, however, that the rejection it receives to its reconnect request is due to protocol requirement, and may not reflect the actual desire of the corresponding process. It should also note that its reconnection request may be re-issued after the first reconnection activity is complete. This is an example of a situation where an option change request can be re-issued after a denial, without a corresponding change in state.

ASIDE:

The usefulness of reconnection is severely limited by its specification as an option for Telnet (i.e. terminal like) connections, rather than as part of a host-host protocol which would allow it to be applied to general connections. First, it is questionable whether most systems will allow a user task to maintain more than one Telnet connection. If not, a process on such a system can not readily initiate a reconnection request.

See RFC 36

Second, there are certain indirect benefits that would result from including reconnection in a host-host protocol. Placing it at that level could simplify some of the timing problems in

establishing the new connection. For example, an NCP would be aware when a reconnection was in progress, and therefore would not need to act as hastily with an RFC for a socket currently in use (i.e. connection still open) but involved in the reconnection. Since it is dealing with another NCP directly, it can expect to receive the "reconnect go ahead" reasonably soon, barring system crash. Also, the information necessary to complete the reconnection subnegotiation is available at the NCP level, whereas it must be duplicately maintained by the Telnet service routine when the potential for reconnection exists.

Note RFC 36 (B)

Finally, the entire notion of reconnection is framed in terms of the entities of host-host protocol. By placing it at a higher level without adequate provision at the host-host level, an artificial and rigid constraint is placed on the type of communication path which may be part of a reconnection. Since host-host protocol is the basis for function oriented levels, the notion of redirecting communication paths certainly is more suited to the semantically uninterpreted realm of OPENing and CLOSEing connections, rather than the realm of "open an 8 bit ASCII path with the conventions that ..."

II. IMPLEMENTATION DETAILS

1. A process initiating a reconnection designates one of the object processes as passive (i.e. to listen for RFCs), and the other as active (i.e. to send RFCs). The reconnection protocol does not specify the assignment of the active/passive roles, so the process is free in its selection. However, information regarding the types of participants in the reconnection attempt may dictate a role selection which will contribute to the eventual successful completion of the reconnection. Ignoring such information could conceivably force cancellation of the attempt. Certain types of hosts (e.g. space limited TIPs) may be better suited for active participation, since it need not go through the procedure of verifying the identity of the sender. The passive process should go through such verification. Other types of hosts (e.g. one whose NCP will not let an arbitrary process listen on a socket) may be better suited for the active role. As more systems implement the reconnection option, the preferences of various types of systems will become known, and more definitive rules may emerge.

2. To avoid posible deadlock, the active (passive) process must simultaneously send (listen for) RFCs for both send and receive connections which will form the new Telnet connection. Since the reconnection protocol does not specify an ordering for establishing the connections, it is important that passive processes listen in parallel on both the potential send and receive sockets, and that active processes send RFCs in parallel for both the potential send and receive sockets.

3. There are two levels of error recovery involved in reconnection. One level is required to handle the conditions where network and system delays cause the attempt to establish the new connection to get out of synchrony (e.g. the RFC arrives at the passive host before the passive process listens), or cause system timeouts. When these conditions occur the sockets/connections should be returned to a state in which the faulting operation can be automatically retried. The second level of recovery involves the failure of all such attempts to establish communication with the active (passive) process. The duration of these attempts may be influenced by such factors as the recovery procedures available, and whether or not a human user is awaiting the outcome. Recovery at this point is difficult since the connections with the initiating process have already been broken. Attempts to connect to some reasonable alternative (perhaps local, perhaps attempting to connect back to the original source of the reconnection) should be initiated if second level error recovery is necessary, indicating complete reconnection failure.

4. A useful addition to the reconnection mechanism would be the definition of a standard way to reestablish contact with the reconnection initiator on task termination (including can´t complete reconnection).

III. TENEX RELATED DETAILS

The context for our experiments was that of a TIP user using a TIPSER/RSEXEC. The TIPSER/RSEXEC would first authenticate the TIP user and then serve as a command interpreter. Among the available commands was one called TELCONN (TELnet CONNect) for connecting to other sites for service. A TELCONN command would trigger an attempt by the TIPSER/RSEXEC to reconnect the "TIP" directly to the host which was the target of the TELCONN request (normally this would usually be a logger process at the host). When the reconnection is completed, the TIP is directly connected to the new job, and the TIPSER/RSEXEC is completely eliminated from the communication path. To avoid programming the TIP, a TENEX process was used to simulate the TIP.

Certain features of TENEX caused problems in creating the desired interaction between the TENEX jobs involved in the reconnection experiment. They are presented here because there may be similar problems in other systems.

1. Along with the features supplied by the TENEX Telnet interface via the ATPTY system call (which transforms a pair of unused network connections into a Telnet connection pair), comes a loss of certain control functions. A program loses the ability to control when data is sent (i.e. loss of the use of the MTOPR system call to force transmission of buffered data), and can no longer determine the remote host/socket for the network connection (i.e.

GDSTS system call). In a highly interactive mode such as option negotiation, short messages remaining in system buffers can result in a deadlock. A process must be able to override the buffering strategy at the conclusion of a logical message. Failure to have access to such a mechanism (e.g. MTOPR) requires that the connection be opened in a non-buffered mode, which is wasteful most of the time. Similarly, the inability to obtain the remote host/socket names of the connection requires that this information be remembered by the program for the duration of the connection in case it is needed. (This is the case despite the fact that the operating system maintains the information in any event. The need to access this information arises when we wish to reconnect the Telnet connection which linked the "TIP" to the TIPSER/RSEXEC).

2. There is no facility in TENEX for handling (initiating or responding to) Telnet options not recognized by the Telnet server. An interface between a user program and the option negotiation mechanism would be useful for testing new options and for implementing privates ones. Lack of this interface can be circumvented by switching the connection to binary mode transmission and reception. This works only if option negotiation is between two user processes (both aware of the binary transmission), since if a user process tried to negotiate with a system Telnet server obeying the binary transmission option, the required doubling of IACs for binary output would cause the request to be misinterpreted at the system Telnet.

3. The switch to binary transmission requires two option negotiations. During this period data transfer is possible. However, the actual data transferred is dependent on the state of the negotiation at that point (e.g. depending upon the state, the IAC character may or may not be doubled). There does not seem to be a facility for alerting the process that the option has been accepted (rejected) and that all further transmissions will be in the new mode (binary). Perhaps suspending the process for the duration of the (timed out) option negotiation would eliminate this period of uncertainty in the mode switch. In TENEX, this could be coupled with pseudo-interrupts to note option negotiation failure for certain critical user initiated options.

4. During peak load conditions, RFCs sent by the operating system (NCP) in response to program requests (OPENF system calls) were frequently timed out by the system. The passive process listening for the RFCs did not get rescheduled quickly enough to reply to the RFCs (acceptance or rejection) before they were timed out by the system. A confusing situation arose because of the difference in initiating the two connections (send and receive) that were to form the full-duplex path between the processes. One OPENF specified immediate return, while the other waited for completion of the RFC. If both requests timed out, the states of the corresponding connections were different, and therefore the retry mechanism had to handle each differently (i.e. the "immediate return" connection had to be closed via CLOSF, whereas the other did

not). This seems to be an unnecessary complication. Also, the frequency of timeout during heavy load conditions may indicate that the RFC timeout interval is too short.

5. In the TENEX user interface to the network there is no concept of logical messages when more than one process (fork) shares a network connection. Telnet option negotiation sequences are examples of strings which must be sent in proper order, without interceeding characters of any nature in order to have correct meaning. Even when a TENEX "string out" (SOUT) operation is executed by a process, which is indicative of some logical relationship between the characters of the string, the transmission is not guaranteed to be free from interference from other processes sending data over the same connection. (Multi-process organization for managing network connections is very common. One process is typically used to handle user output to the network, while another process reads data from the network and replies as required by protocol to certain network input). These processes must synchronize on every output (and input) to assure the logical integrity of their messages. This synchronization would seem to be more suitably handled by the system routines which manage network connections and handle string I/O.